SQL Assignment

```
In [1]: import pandas as pd
        import sqlite3

        from IPython.display import display, HTML
```

```
In [ ]: # Note that this is not the same db we have used in course videos, please download from this link
        # https://drive.google.com/file/d/1O-1-L1DdNxEK6O6nG2jS31MbrMh-OnXM/view?usp=sharing
```

```
In [2]: from google.colab import drive
        drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
In [3]: !ls '/content/drive/MyDrive/3_SQL/'
```

```
 Db-IMDB-Assignment.db   sql_question.gdoc   'SQL sample queries.ipynb'
 db_schema.jpeg          sql_question.pdf
```

```
In [4]: conn = sqlite3.connect('/content/drive/MyDrive/3_SQL/Db-IMDB-Assignment.db')
```

Overview of all tables

```
In [5]: tables = pd.read_sql_query("SELECT NAME AS 'Table_Name' FROM sqlite_master WHERE type='table'",conn)
        tables = tables["Table_Name"].values.tolist()
```

```
In [6]: tables
```

```
Out[6]: ['Movie',
         'Genre',
         'Language',
         'Country',
         'Location',
         'M_Location',
         'M_Country',
         'M_Language',
         'M_Genre',
         'Person',
         'M_Producer',
         'M_Director',
         'M_Cast']
```

```
In [7]: for table in tables:
            query = "PRAGMA TABLE_INFO({})".format(table)
            schema = pd.read_sql_query(query,conn)
            print("Schema of",table)
            display(schema)
            print("-"*100)
            print("\n")
```

Schema of Movie

	cid	name	type	notnull	dflt_value	pk
0	0	index	INTEGER	0	None	0
1	1	MID	TEXT	0	None	0
2	2	title	TEXT	0	None	0
3	3	year	TEXT	0	None	0
4	4	rating	REAL	0	None	0
5	5	num_votes	INTEGER	0	None	0

Schema of Genre

	cid	name	type	notnull	dflt_value	pk
0	0	index	INTEGER	0	None	0
1	1	Name	TEXT	0	None	0
2	2	GID	INTEGER	0	None	0

Schema of Language

	cid	name	type	notnull	dflt_value	pk
0	0	index	INTEGER	0	None	0
1	1	Name	TEXT	0	None	0
2	2	LAID	INTEGER	0	None	0

Schema of Country

	cid	name	type	notnull	dflt_value	pk
0	0	index	INTEGER	0	None	0
1	1	Name	TEXT	0	None	0
2	2	CID	INTEGER	0	None	0

Schema of Location

	cid	name	type	notnull	dflt_value	pk
0	0	index	INTEGER	0	None	0
1	1	Name	TEXT	0	None	0
2	2	LID	INTEGER	0	None	0

Schema of M_Location

	cid	name	type	notnull	dflt_value	pk
0	0	index	INTEGER	0	None	0
1	1	MID	TEXT	0	None	0
2	2	LID	REAL	0	None	0
3	3	ID	INTEGER	0	None	0

Schema of M_Country

	cid	name	type	notnull	dflt_value	pk
0	0	index	INTEGER	0	None	0
1	1	MID	TEXT	0	None	0
2	2	CID	REAL	0	None	0
3	3	ID	INTEGER	0	None	0

Schema of M_Language

	cid	name	type	notnull	dflt_value	pk
0	0	index	INTEGER	0	None	0
1	1	MID	TEXT	0	None	0
2	2	LAID	INTEGER	0	None	0
3	3	ID	INTEGER	0	None	0

Schema of M_Genre

	cid	name	type	notnull	dflt_value	pk
0	0	index	INTEGER	0	None	0
1	1	MID	TEXT	0	None	0
2	2	GID	INTEGER	0	None	0
3	3	ID	INTEGER	0	None	0

Schema of Person

	cid	name	type	notnull	dflt_value	pk
0	0	index	INTEGER	0	None	0
1	1	PID	TEXT	0	None	0
2	2	Name	TEXT	0	None	0
3	3	Gender	TEXT	0	None	0

Schema of M_Producer

	cid	name	type	notnull	dflt_value	pk
0	0	index	INTEGER	0	None	0
1	1	MID	TEXT	0	None	0
2	2	PID	TEXT	0	None	0
3	3	ID	INTEGER	0	None	0

Schema of M_Director

	cid	name	type	notnull	dflt_value	pk
0	0	index	INTEGER	0	None	0
1	1	MID	TEXT	0	None	0
2	2	PID	TEXT	0	None	0
3	3	ID	INTEGER	0	None	0

Schema of M_Cast

	cid	name	type	notnull	dflt_value	pk
0	0	index	INTEGER	0	None	0
1	1	MID	TEXT	0	None	0
2	2	PID	TEXT	0	None	0
3	3	ID	INTEGER	0	None	0

Useful tips:

1. the year column in 'Movie' table, will have few chracters other than numbers which you need to be preprocessed, you need to get a substring of last 4 characters, its better if you convert it as int type, ex: CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER)
2. For almost all the TEXT columns we have show, please try to remove trailing spaces, you need to use TRIM() function
3. When you are doing count(coulmn) it won't consider the "NULL" values, you might need to explore other alternatives like Count(*)

Q1 --- List all the directors who directed a 'Comedy' movie in a leap year. (You need to check that the genre is 'Comedy' and year is a leap year) Your query should return director name, the movie name, and the year.

To determine whether a year is a leap year, follow these steps:

- **STEP-1:** If the year is evenly divisible by 4, go to step 2. Otherwise, go to step 5.
- **STEP-2:** If the year is evenly divisible by 100, go to step 3. Otherwise, go to step 4.
- **STEP-3:** If the year is evenly divisible by 400, go to step 4. Otherwise, go to step 5.
- **STEP-4:** The year is a leap year (it has 366 days).
- **STEP-5:** The year is not a leap year (it has 365 days).

Year 1900 is divisible by 4 and 100 but it is not divisible by 400, so it is not a leap year.

```
In [8]: %%time
        def grader_1(q1):
            q1_results  = pd.read_sql_query(q1,conn)
            print(q1_results.head(10))
            assert (q1_results.shape == (232,3))

        query1 ='''Select Distinct trim(P.Name) as Director, M.title as Movie, CAST(SUBSTR(TRIM(M.year),-4) AS
         INT) as year From Movie M
                                Join M_Director MD on M.MID = MD.MID
                                Join Person P on trim(MD.PID) = P.PID
                                Join M_Genre MG on M.MID = MG.MID
                                Join Genre G on MG.GID = G.GID

                            WHERE trim(G.Name) LIKE '%Comedy%'
                                AND (((CAST(SUBSTR(TRIM(M.year),-4) AS INT) % 4 = 0)
                                AND (CAST(SUBSTR(TRIM(M.year),-4) AS INT)%100 != 0))
                                OR (CAST(SUBSTR(TRIM(M.year),-4) AS INT)%400 = 0))
                            ORDER BY year'''

        grader_1(query1)
```

```
          Director            Movie  year
0      Amit Mitra       Jagte Raho  1956
1    Chetan Anand         Funtoosh  1956
2     Satyen Bose          Jagriti  1956
3     Mohan Segal        New Delhi  1956
4      S.U. Sunny         Kohinoor  1960
5       Bimal Roy           Parakh  1960
6     R.K. Nayyar    Love in Simla  1960
7      K. Shankar         Rajkumar  1964
8  Shakti Samanta  Kashmir Ki Kali  1964
9   Ram Mukherjee           Leader  1964
CPU times: user 72.6 ms, sys: 2.96 ms, total: 75.5 ms
Wall time: 82.9 ms
```

Q2 --- List the names of all the actors who played in the movie 'Anand' (1971)

```
In [9]: %%time
        def grader_2(q2):
            q2_results  = pd.read_sql_query(q2,conn)
            print(q2_results.head(10))
            assert (q2_results.shape == (17,1))


        query2 ='''SELECT DISTINCT trim(P.Name) AS Actor_Names FROM Movie M
                                    Join M_Cast MC on M.MID = MC.MID
                                    Join Person P on trim(MC.PID) = P.PID
                                    WHERE M.title = 'Anand' AND M.year = 1971 '''

        grader_2(query2)
```

```
       Actor_Names
0  Amitabh Bachchan
1     Rajesh Khanna
2    Brahm Bhardwaj
3        Ramesh Deo
4         Seema Deo
5        Dev Kishan
6       Durga Khote
7     Lalita Kumari
8      Lalita Pawar
9      Atam Prakash
CPU times: user 154 ms, sys: 5.23 ms, total: 159 ms
Wall time: 176 ms
```

Q3 --- List all the actors who acted in a film before 1970 and in a film after 1990. (That is: < 1970 and > 1990.)

```
In [10]: %%time

         def grader_3a(query_less_1970, query_more_1990):
             q3_a = pd.read_sql_query(query_less_1970,conn)
             q3_b = pd.read_sql_query(query_more_1990,conn)
             return (q3_a.shape == (4942,1)) and (q3_b.shape == (62570,1))

         query_less_1970 ='''SELECT trim(P.PID) as id FROM Movie M
                                     Join M_Cast MC on M.MID = MC.MID
                                     Join Person P on trim(MC.PID) = P.PID
                                     WHERE CAST(SUBSTR(M.year,-4) AS INT) < 1970 '''

         query_more_1990 ='''SELECT trim(P.PID) as id FROM Movie M
                                     Join M_Cast MC on M.MID = MC.MID
                                     Join Person P on trim(MC.PID) = P.PID
                                     WHERE CAST(SUBSTR(M.year,-4) AS INT) > 1990'''

         print(grader_3a(query_less_1970, query_more_1990))

         # using the above two queries, you can find the answer to the given question
```

```
True
CPU times: user 404 ms, sys: 16.7 ms, total: 421 ms
Wall time: 428 ms
```

```
In [11]: %%time
         def grader_3(q3):
             q3_results  = pd.read_sql_query(q3,conn)
             print(q3_results.head(10))
             assert (q3_results.shape == (300,1))

         query3 = """ SELECT DISTINCT p.Name FROM Movie m
                         JOIN M_Cast mc ON m.MID = mc.MID
                         JOIN Person p ON trim(mc.PID) = p.PID
                         WHERE CAST(SUBSTR(m.year,-4) AS INT) < 1970
                      AND trim(p.PID) IN
                          (SELECT DISTINCT trim(p.PID) as pid FROM Movie m
                                JOIN M_Cast mc ON m.MID = mc.MID
                                JOIN Person p ON trim(mc.PID) = p.PID
                                WHERE CAST(SUBSTR(m.year,-4) AS INT) > 1990)
                      GROUP BY trim(p.PID)
                 """
         grader_3(query3)
```

```
                 Name
0    Amitabh Bachchan
1  Mohandas K. Gandhi
2               Rekha
3          Dharmendra
4    Prithviraj Kapoor
5       Shammi Kapoor
6       Shashi Kapoor
7       Rajesh Khanna
8         Hema Malini
9         Sanjay Dutt
CPU times: user 430 ms, sys: 11.7 ms, total: 441 ms
Wall time: 447 ms
```

Q4 --- List all directors who directed 10 movies or more, in descending order of the number of movies they directed. Return the directors' names and the number of movies each of them directed.

```
In [12]: %%time

         def grader_4a(query_4a):
             query_4a = pd.read_sql_query(query_4a,conn)
             print(query_4a.head(10))
             return (query_4a.shape == (1462,2))

         query_4a =''' SELECT DISTINCT trim(md.PID) AS Director_ID, COUNT(md.PID) AS Movie_Count FROM M_Director
          md
                                       JOIN Person p on trim(md.PID) = p.PID
                                       GROUP BY trim(MD.PID) '''
         print(grader_4a(query_4a))

         # using the above query, you can write the answer to the given question
```

```
  Director_ID  Movie_Count
0   nm0000180            1
1   nm0000187            1
2   nm0000229            1
3   nm0000269            1
4   nm0000386            1
5   nm0000487            2
6   nm0000965            1
7   nm0001060            1
8   nm0001162            1
9   nm0001241            1
True
CPU times: user 61.2 ms, sys: 699 µs, total: 61.9 ms
Wall time: 67.8 ms
```

```
In [13]: %%time
         def grader_4(q4):
             q4_results  = pd.read_sql_query(q4,conn)
             print(q4_results.head(10))
             assert (q4_results.shape == (58,2))

         query4 = '''SELECT DISTINCT trim(p.Name) AS Director_Name, COUNT(md.PID) AS Movie_Count FROM M_Director
          md
                                       JOIN Person p on trim(md.PID) = p.PID
                                       GROUP BY trim(md.PID) HAVING Movie_Count>=10 ORDER BY Movie_Count DESC'''
         grader_4(query4)
```

```
          Director_Name  Movie_Count
0          David Dhawan           39
1          Mahesh Bhatt           35
2          Priyadarshan           30
3        Ram Gopal Varma          30
4          Vikram Bhatt           29
5  Hrishikesh Mukherjee           27
6            Yash Chopra          21
7       Basu Chatterjee           19
8        Shakti Samanta           19
9          Subhash Ghai           18
CPU times: user 65.2 ms, sys: 0 ns, total: 65.2 ms
Wall time: 72.5 ms
```

Q5.a --- For each year, count the number of movies in that year that had only female actors.

```
In [14]: %%time

         # note that you don't need TRIM for person table

         def grader_5aa(query_5aa):
             query_5aa = pd.read_sql_query(query_5aa,conn)
             print(query_5aa.head(10))
             return (query_5aa.shape == (8846,3))

         query_5aa =''' SELECT mc.MID, p.Gender, COUNT(1) FROM M_Cast mc
                                     JOIN Person p on trim(mc.PID) = p.PID
                                     GROUP BY mc.MID, p.Gender '''

         print(grader_5aa(query_5aa))

         def grader_5ab(query_5ab):
             query_5ab = pd.read_sql_query(query_5ab,conn)
             print(query_5ab.head(10))
             return (query_5ab.shape == (3469, 3))

         query_5ab =''' SELECT Distinct mc.MID, p.Gender, COUNT(1) FROM M_Cast mc
                                     JOIN Person p on trim(mc.PID) = p.PID
                                     WHERE (p.Gender = 'Male')
                                     GROUP BY mc.MID, p.Gender '''


         print(grader_5ab(query_5ab))


         # using the above queries, you can write the answer to the given question
```

```
         MID  Gender  COUNT(1)
0  tt0021594    None         1
1  tt0021594  Female         3
2  tt0021594    Male         5
3  tt0026274    None         2
4  tt0026274  Female        11
5  tt0026274    Male         9
6  tt0027256    None         2
7  tt0027256  Female         5
8  tt0027256    Male         8
9  tt0028217  Female         3
True
         MID Gender  COUNT(1)
0  tt0021594   Male         5
1  tt0026274   Male         9
2  tt0027256   Male         8
3  tt0028217   Male         7
4  tt0031580   Male        27
5  tt0033616   Male        46
6  tt0036077   Male        11
7  tt0038491   Male         7
8  tt0039654   Male         6
9  tt0040067   Male        10
True
CPU times: user 339 ms, sys: 5.41 ms, total: 344 ms
Wall time: 346 ms
```

```
In [15]: %%time
         def grader_5a(q5a):
             q5a_results  = pd.read_sql_query(q5a,conn)
             print(q5a_results.head(10))
             assert (q5a_results.shape == (4,2))

         query5a ='''SELECT CAST(SUBSTR(TRIM(m.year),-4) AS INT) AS YEAR, COUNT(m.year) AS Female_Cast_Only_Movi
         es FROM Movie m
                                     WHERE m.MID NOT IN
                                     (SELECT mc.MID FROM M_Cast mc
                                     JOIN Person p on trim(mc.PID) = p.PID
                                     Where p.Gender = 'Male')
                                     GROUP BY YEAR
                                     ORDER BY Female_Cast_Only_Movies'''

         grader_5a(query5a)
```

```
   YEAR  Female_Cast_Only_Movies
0  1939                        1
1  1999                        1
2  2000                        1
3  2018                        1
CPU times: user 146 ms, sys: 200 µs, total: 146 ms
Wall time: 152 ms
```

Q5.b --- Now include a small change: report for each year the percentage of movies in that year with only female actors, and the total number of movies made that year. For example, one answer will be: 1990 31.81 13522 meaning that in 1990 there were 13,522 movies, and 31.81% had only female actors. You do not need to round your answer.

```
In [16]: # https://stackoverflow.com/questions/11719044/how-to-get-a-float-result-by-dividing-two-integer-values
         -using-t-sql

         %%time
         def grader_5b(q5b):
             q5b_results  = pd.read_sql_query(q5b,conn)
             print(q5b_results.head(10))
             assert (q5b_results.shape == (4,3))

         query5b = '''SELECT yr_female.YEAR, cast(Female_Cast_Only_Movies AS float)/cast(TOTAL_MOVIES AS float)
          AS Percentage_Female_Only_Movie, TOTAL_MOVIES
                                     FROM (
                                       SELECT CAST(SUBSTR(TRIM(m.year),-4) AS INT) AS YEAR, cast(COUNT(m.year) a
         s INT) AS Female_Cast_Only_Movies FROM Movie m
                                       WHERE m.MID NOT IN (
                                         SELECT mc.MID FROM M_Cast mc
                                         JOIN Person p on trim(mc.PID) = p.PID
                                         Where p.Gender = 'Male'
                                         )
                                         GROUP BY YEAR
                                         ) yr_female
                                       JOIN
                                         (
                                           Select CAST(SUBSTR(year,-4) AS INT) AS YEAR, COUNT(TRIM(MID)) TOTAL_M
         OVIES FROM Movie
                                           GROUP BY CAST(SUBSTR(year,-4) AS INT)
                                         ) total_movie
                                       ON total_movie.YEAR = yr_female.YEAR
                                       '''
         grader_5b(query5b)
```

```
   YEAR  Percentage_Female_Only_Movie  TOTAL_MOVIES
0  1939                      0.500000             2
1  1999                      0.015152            66
2  2000                      0.015625            64
3  2018                      0.009615           104
CPU times: user 147 ms, sys: 3.49 ms, total: 151 ms
Wall time: 153 ms
```

Q6 --- Find the film(s) with the largest cast. Return the movie title and the size of the cast. By "cast size" we mean the number of distinct actors that played in that movie: if an actor played multiple roles, or if it simply occurs multiple times in casts, we still count her/him only once.

```
In [17]: %%time
         def grader_6(q6):
             q6_results  = pd.read_sql_query(q6,conn)
             print(q6_results.head(10))
             assert (q6_results.shape == (3473, 2))

         query6 = ''' SELECT m.title as Title, COUNT(mc.PID) AS Cast_Count FROM MOVIE m
                                         JOIN M_Cast mc on M.MID = mc.MID
                                         GROUP BY mc.MID ORDER BY Cast_Count DESC '''
         grader_6(query6)
```

```
                         Title  Cast_Count
0                Ocean's Eight         238
1                     Apaharan         233
2                         Gold         215
3              My Name Is Khan         213
4  Captain America: Civil War         191
5                     Geostorm         170
6                      Striker         165
7                         2012         154
8                       Pixels         144
9        Yamla Pagla Deewana 2         140
CPU times: user 161 ms, sys: 15.9 ms, total: 177 ms
Wall time: 179 ms
```

Q7 --- A decade is a sequence of 10 consecutive years.

For example, say in your database you have movie information starting from 1931.

the first decade is 1931, 1932, ..., 1940,

the second decade is 1932, 1933, ..., 1941 and so on.

Find the decade D with the largest number of films and the total number of films in D.

```
In [18]: # https://stackoverflow.com/questions/51609285/query-for-find-the-decade-with-the-largest-number-of-rec
         ords

         %%time
         def grader_7a(q7a):
             q7a_results  = pd.read_sql_query(q7a,conn)
             print(q7a_results.head(10))
             assert (q7a_results.shape == (78, 2))

         query7a = """ Select CAST(SUBSTR(year,-4) AS INT) AS YEAR, COUNT(TRIM(MID)) TOTAL_MOVIES FROM Movie
                         GROUP BY CAST(SUBSTR(year,-4) AS INT) ORDER BY YEAR """
         grader_7a(query7a)

         # using the above query, you can write the answer to the given question
```

```
   YEAR  TOTAL_MOVIES
0  1931             1
1  1936             3
2  1939             2
3  1941             1
4  1943             1
5  1946             2
6  1947             2
7  1948             3
8  1949             3
9  1950             2
CPU times: user 10.1 ms, sys: 1.14 ms, total: 11.2 ms
Wall time: 15.8 ms
```

```
In [19]: %%time
         def grader_7b(q7b):
             q7b_results  = pd.read_sql_query(q7b,conn)
             print(q7b_results.head(10))
             assert (q7b_results.shape == (713, 4))

         query7b = """
             SELECT t1.YEAR AS Movie_Year, t1.TOTAL_MOVIES AS Total_Movies, t2.YEAR AS Movie_Year, t2.TOTAL_MOVI
         ES AS Total_Movies
               FROM
                 (Select CAST(SUBSTR(year,-4) AS INT) AS YEAR, COUNT(TRIM(MID)) TOTAL_MOVIES FROM Movie GROUP BY
          CAST(SUBSTR(year,-4) AS INT) ORDER BY YEAR) t1
                 JOIN
                 (Select CAST(SUBSTR(year,-4) AS INT) AS YEAR, COUNT(TRIM(MID)) TOTAL_MOVIES FROM Movie GROUP BY
          CAST(SUBSTR(year,-4) AS INT) ORDER BY YEAR)  t2

                 ON t1.YEAR <= (t2.YEAR+9) AND t2.YEAR <= (t1.YEAR+9) AND t2.YEAR >= t1.YEAR
                 """
         grader_7b(query7b)
         # if you see the below results the first movie year is less than 2nd movie year and
         # 2nd movie year is less or equal to the first movie year+9

         # using the above query, you can write the answer to the given question
```

```
   Movie_Year  Total_Movies  Movie_Year  Total_Movies
0        1931             1        1931             1
1        1931             1        1936             3
2        1931             1        1939             2
3        1936             3        1936             3
4        1936             3        1939             2
5        1936             3        1941             1
6        1936             3        1943             1
7        1939             2        1939             2
8        1939             2        1941             1
9        1939             2        1943             1
CPU times: user 17.6 ms, sys: 0 ns, total: 17.6 ms
Wall time: 20.2 ms
```

```
In [20]: # https://stackoverflow.com/questions/51609285/query-for-find-the-decade-with-the-largest-number-of-rec
         ords

         %%time
         def grader_7(q7):
             q7_results  = pd.read_sql_query(q7,conn)
             print(q7_results.head(10))
             assert (q7_results.shape == (1, 2))

         query7 = """ SELECT COUNT(TRIM(MID)) as Decade_Movie_Count, t.year as Decade_Start
                       FROM
                       (SELECT DISTINCT year from Movie) t
                           JOIN Movie m
                             on CAST(SUBSTR(m.year,-4) AS INT) >= t.year and CAST(SUBSTR(m.year,-4) AS INT) <=
          t.year + 9
                             GROUP BY t.year
                             ORDER BY COUNT(TRIM(MID)) DESC
                             limit 1"""

         grader_7(query7)
         # if you check the output we are printinng all the year in that decade, its fine you can print 2008 or
          2008-2017
```

```
   Decade_Movie_Count Decade_Start
0                1203         2008
CPU times: user 122 ms, sys: 0 ns, total: 122 ms
Wall time: 129 ms
```

Q8 --- Find all the actors that made more movies with Yash Chopra than any other director.

```
In [21]: %%time
         def grader_8a(q8a):
             q8a_results  = pd.read_sql_query(q8a,conn)
             print(q8a_results.head(10))
             assert (q8a_results.shape == (73408, 3))

         # Refer--> https://stackoverflow.com/questions/57743908/sql-query-to-find-an-actors-who-did-more-films-
         with-quentin-tarantino

         query8a = """ SELECT cast_id, director_id, movies_count FROM
                           (SELECT md.PID director_id, mc.PID cast_id, COUNT(*) movies_count FROM M_Director md
                           JOIN M_Cast mc ON md.MID = mc.MID
                           GROUP BY cast_id, director_id) """
         grader_8a(query8a)

         # using the above query, you can write the answer to the given question
```

```
     cast_id director_id  movies_count
0  nm0000002   nm0496746             1
1  nm0000027   nm0000180             1
2  nm0000039   nm0896533             1
3  nm0000042   nm0896533             1
4  nm0000047   nm0004292             1
5  nm0000073   nm0485943             1
6  nm0000076   nm0000229             1
7  nm0000092   nm0178997             1
8  nm0000093   nm0000269             1
9  nm0000096   nm0113819             1
CPU times: user 327 ms, sys: 15.2 ms, total: 342 ms
Wall time: 344 ms
```

Pseudo Code:

```
select the names of actors and movie_count
          (select actor_pid,direct_pid,movie_count from

            (select actor_pid, dicrector_pid, movie_count and grouping by actor_pid, dicrector_p
id)-> 8a

          where (actor_pid, movie_count) in

            (select actor_pid,max(movie_count) from
                (select actor_pid, dicrector_pid, movie_count and grouping by actor_pid, dicrect
or_pid)-> 8a
                group by actor_pid)
          and director is yash)
```

```
In [22]: %%time

         def grader_8(q8):
             q8_results  = pd.read_sql_query(q8,conn)
             print(q8_results.head(10))
             print(q8_results.shape)
             assert (q8_results.shape == (245, 2))

         query8 = """ SELECT p.Name, a.movies_count FROM Person p,
                         (SELECT cast_id, director_id, movies_count FROM
                           (SELECT md.PID director_id, mc.PID cast_id, COUNT(*) movies_count FROM M_Director md
                           JOIN M_Cast mc ON md.MID = mc.MID
                           GROUP BY cast_id, director_id)
                           WHERE (cast_id, movies_count) IN
                               (SELECT cast_id, max(movies_count) FROM
                                   (SELECT md.PID director_id, mc.PID cast_id, Count(*) movies_count FROM M_Dire
         ctor md
                                   JOIN M_Cast mc ON md.MID = mc.MID
                                   GROUP BY cast_id, director_id)
                                   GROUP BY Cast_id)
                             AND TRIM(director_id) = 'nm0007181') a
                             WHERE TRIM(a.cast_id) = TRIM(p.PID) ORDER BY movies_count DESC """

         grader_8(query8)
```

```
                Name  movies_count
0        Jagdish Raj            11
1   Manmohan Krishna            10
2           Iftekhar             9
3      Shashi Kapoor             7
4      Rakhee Gulzar             5
5     Waheeda Rehman             5
6           Ravikant             4
7     Achala Sachdev             4
8        Neetu Singh             4
9      Leela Chitnis             3
(245, 2)
CPU times: user 2.14 s, sys: 11.6 ms, total: 2.15 s
Wall time: 2.16 s
```

Q9 --- The Shahrukh number of an actor is the length of the shortest path between the actor and Shahrukh Khan in the "co-acting" graph. That is, Shahrukh Khan has Shahrukh number 0; all actors who acted in the same film as Shahrukh have Shahrukh number 1; all actors who acted in the same film as some actor with Shahrukh number 1 have Shahrukh number 2, etc. Return all actors whose Shahrukh number is 2.

ref: https://mail.google.com/mail/u/0?ui=2&ik=4c6e313f68&attid=0.1&permmsgid=msg-f:1687069395112113920&th=1769acad27173700&view=fimg&sz=s0-l75-ft&attbid=ANGjdJ9XzKau2f1jB3GvLyT8f2Co1P6VJbexhAhGHtvblGZSwMkesV4I7KaNLqc5oTfSS2QR56ekYrtoTRENm6Ef-l230tN6qwQsDiroGfKB25KK8CqM7KFS9BImPCs&disp=emb&realattid=ii_kfqlznll0

In [23]:

```
%%time
def grader_9a(q9a):
    q9a_results  = pd.read_sql_query(q9a,conn)
    print(q9a_results.head(10))
    print(q9a_results.shape)
    assert (q9a_results.shape == (2382, 1))

query9a = """ SELECT DISTINCT TRIM(PID) AS S1_PID FROM M_Cast WHERE TRIM(PID) IN
                (SELECT DISTINCT TRIM(PID) FROM M_Cast WHERE TRIM(MID) IN
                   (SELECT DISTINCT TRIM(MID) FROM M_Cast WHERE TRIM(PID) IN
                      (SELECT TRIM(PID) FROM Person WHERE TRIM(Name) Like '%Shah Rukh Khan%')))
                   AND TRIM(PID) NOT IN
                      (SELECT TRIM(PID) FROM Person WHERE TRIM(Name) Like '%Shah Rukh Khan%')
                      ORDER BY TRIM(PID)
                    """
grader_9a(query9a)
# using the above query, you can write the answer to the given question

# selecting actors who acted with srk (S1)
# selecting all movies where S1 actors acted, this forms S2 movies list
# selecting all actors who acted in S2 movies, this gives us S2 actors along with S1 actors
# removing S1 actors from the combined list of S1 & S2 actors, so that we get only S2 actors
```

```
      S1_PID
0  nm0000818
1  nm0000821
2  nm0001934
3  nm0002043
4  nm0004109
5  nm0004334
6  nm0004335
7  nm0004363
8  nm0004418
9  nm0004429
(2382, 1)
CPU times: user 101 ms, sys: 10 ms, total: 111 ms
Wall time: 117 ms
```

In [24]:

```
%%time
def grader_9(q9):
    q9_results  = pd.read_sql_query(q9,conn)
    print(q9_results.head(10))
    print(q9_results.shape)
    assert (q9_results.shape == (25698, 1))

query9 = """ SELECT  Name AS Actor_Names FROM Person WHERE TRIM(PID) IN
              (SELECT DISTINCT TRIM(PID) FROM M_Cast WHERE MID IN
                (SELECT DISTINCT MID FROM M_Cast WHERE TRIM(PID) IN
                  (SELECT DISTINCT TRIM(PID) FROM M_Cast WHERE TRIM(MID) IN
                    (SELECT DISTINCT MID FROM M_Cast WHERE TRIM(PID) IN
                      (SELECT DISTINCT TRIM(PID) FROM Person WHERE TRIM(Name) Like '%Shah Rukh Kha
n%')))))
                AND TRIM(PID) NOT IN
                     (SELECT DISTINCT TRIM(PID) FROM M_Cast WHERE MID IN
                        (SELECT DISTINCT MID FROM M_Cast WHERE TRIM(PID) IN
                          (SELECT DISTINCT TRIM(PID) FROM Person WHERE TRIM(Name) Like '%Shah Rukh Kha
n%')))
         """
grader_9(query9)
```

```
              Actor_Names
0            Freida Pinto
1             Rohan Chand
2            Damian Young
3         Waris Ahluwalia
4    Caroline Christl Long
5           Rajeev Pahuja
6       Michelle Santiago
7         Alicia Vikander
8            Dominic West
9          Walton Goggins
(25698, 1)
CPU times: user 294 ms, sys: 12.6 ms, total: 307 ms
Wall time: 316 ms
```